Amortized Analysis

It is an analysis technique which influences the design techniques.


\[ A: 5 4 3 2 1 0 \]

\[
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 \\
\end{array}
\]

\[ n=0 \quad n=\sum_{i=0}^{k} A[i] \quad 2 \]

We increment the number stored in A by the following algorithm.

```
procedure Increment
i:=0
while A[i]=1 do
    A[i]:=0; i:=i+1
endwhile
A[i]:=1
```

Q: What is the total time/number of steps if we increment each from 0 to n-1?

A has \( I + \lceil \log n \rceil \) positions to check for each step. So total \((n \log n)\) is straightforward analysis.
Aggregate Method. This takes a global view rather than counting per operation.

Define \( b_i = \# 1's \) in the binary representation of \( i \).
\[ t_i = \# \text{trailing 1's in the binary representation of } i. \]

The time is proportional to the number of bit changes which is
\[
\sum_{i=0}^{n-2} 1 + t_i \leq n + \frac{n}{2} + \frac{n}{4} + \ldots + 1 \leq 2n
\]

So the total cost is \( T(n) = O(n) \), the amortized cost per operation is
\[
\frac{T(n)}{n} = O(1).
\]

Accounting Method. This analysis charges each operation an "amortized cost".

- If the amortized cost exceeds the actual cost, the excess remains with a data structure as credit.
- If the amortized cost is small enough so that actual cost cannot be covered, it is paid by the credit.

- We define an amortized cost of changing 0 to 1 as $2$
  
  ,, 1 to 0 as $0$

- When $0 \rightarrow 1$, the $1$ covers the actual cost and the other $1$ stays with the bit which is 1. This credit pays for the change later $\Rightarrow 1 \rightarrow 0$.

- Each increment has amortized cost 2. So there are at most $2n$ bit changes.
**Potential Method.** Very similar to accounting method, only difference is that no explicit credit is saved. Instead the credit expressed by a "potential" of the data structure involved.

\[ C_i = \text{Actual cost of the } i\text{th operation} \]
\[ D_i = \text{data structure after the } i\text{th operation} \]
\[ \phi(D_i) = \text{potential of } D_i \]
\[ a_i = C_i + \phi(D_i) - \phi(D_{i-1}) \]

The amortized cost of \( i \)th oprn.

\[
\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} (C_i + \phi(D_i) - \phi(D_{i-1})) = \sum_{i=1}^{n} C_i + \phi(D_n) - \phi(D_0).
\]

If we choose \( \phi \) so that \( \phi(D_0) = 0 \) and \( \phi(D_n) = 0 \) then

\[
\frac{\sum_{i=1}^{n} C_i}{n} \leq \frac{\sum_{i=1}^{n} a_i}{n} \quad \text{Amortized cost is an upper bound to actual cost (total)}
\]
Let us apply the potential method to the binary counting.

Define
\[ \Phi(D_i) = \phi_i \quad \ldots \quad \Phi(D_i) - \Phi(D_{i-1}) = b_i - b_{i-1} \]
\[ = (b_{i-1} - t_{i-1} + 1) - b_{i-1} \]
\[ = 1 - t_{i-1}. \]

Let \( c_i = t_{i-1} + 1 \)

Then \( a_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 2 \)

We have \( \Phi(D_0) = 0 \) and \( \Phi(D_n) \geq 0 \) as desired.

Therefore, \( \sum_{i=1}^{n} a_i = 2n \) is an upper bound on the number of bit changes.